Working With Classes

# Introduction

This document will review information about classes. We have learned that classes are simply just groups of functions. In this document we will review classes more in-depth, specifically looking at how to use objects and how to set-up a data class. Objects allow us to use a class with many different data inputs. Data classes allow us to do extra processing and check with data before we use it in the rest of the program.

# Classes

A class is simple a group of functions. The functions can do various things. It is best practice to separate the script into classes based on their functions. A standard way to set-up functions by setting up three classes; data, processing, and presentation. When working with classes, you can work with them indirectly or directly.

## Object vs Class

When working with a class directly there is just one instance that the set of functions (class) can be stored at one time. However, if we use the code indirectly, we can create many different instances using the same function. We can use one class to do work with many different sets of data or inputs.

The way I thought about it was with cookies. I have a set of cookie cutters that is going to be my class. The cookie cutters can be used over and over. Next, I want to make more than one set of cookies, but with different cookie doughs. The cookies are all the different objects, different inputs to be used in all the different cookie cutters.

I found a few good additional resources to help define the differences between objects and classes. The first one was from a site called LearnPython.org (External Source: [LearnPython](https://www.learnpython.org/en/Classes_and_Objects)). I really liked this source because it stepped through using different inputs but getting different outputs. It really helped figure out the difference between indirect and direct use of classes. ProgramWiz had some other great examples to set through how to set-up a class for indirect use (External Source: [ProgramWiz](https://www.programiz.com/python-programming/class#:~:text=Python%20is%20an%20object%20oriented%20programming%20language.%20Unlike,a%20class%20is%20a%20blueprint%20for%20that%20object.)).

One key difference is the use of “self”. When using the class directly or in static mode, there is not need for the use of “self”. When working indirectly with a class using objects, there is a need for self. You must tell Python that the object is it’s self and not something else. Each object is itself, and Python must know that when working with objects.

## Setting Up a Class

A class is made up of a standard pattern. The pattern involves fields, constructors, properties, and methods.

Text

Description automatically generated with low confidence

Figure 1. Example of the standard pattern used for classes. A class includes, fields, constructor (attributes), properties, and methods.

Fields in classes are just data, such as a product name or product price. Constructors are methods or functions, that run when you create the object from the class. Normally a constructor is used to initialize the value from the field data. Attributes are used as an imaginary or invisible field that is created. See below for an example of a constructor and attributes.

Graphical user interface, text, application

Description automatically generated

Figure 2. Example of how-to set-up a constructor and it’s attributes for product name and product price. It is possible to raise an exception in the error handling process, but when this is done, it causes the program to stop.

Properties are functions that allow you to set different information about the field. The two different properties are a setter and a getter. A setter allows for validation and error handling. A getter allows you to format the attribute or field. There is a set code used when writing the script for a getter and setter, see below. The getter and setter for product name.

Graphical user interface, text, application

Description automatically generated

Figure 3 . Getter and setter code for the product name attribute.

Methods are just functions inside of a class. The functions can be printing a string to reading data to a file. One important not for methods is to know if they are indirect or direct. As mentioned previously, indirect methods require the use of self and direct methods must be defined as static method.

# Working through Assignment 08

The goal for this week assignment was to use indirect objects to create a list of product prices and product names. As with previous assignments we also needed to be able to save the list to a file and then read the file back to the list. I broke the script writing into four parts: data, processing, presentation, and main body.

## Data Class

As discussed in the previous section, the data class is the group of function used to process the data. There are a few standard parts for a data class: fields, constructors and attributes, properties, and methods. For the assignment, we needed to work with the product name and product price.

For product name, I knew that it needed to be a string. The product price needed to be a float. The data class allows for the input values from the user to be checked to make sure that they are the types that we want. We check the data types in the constructors/attributes, and in properties. Properties getters and setters are also another place where the data is checked to make sure it fits what we need.

Graphical user interface, text, application

Description automatically generated

Figure 4. Example of the constructor in my data class for Assignment 08.

The data class is used when first reading the data from the file and when the user enters data into the program. All the data must follow the rules lined out in the data class.

I tested the data class by creating objects for the code to use. In the testing process, I used different data types to see what errors would come up from inputting the “wrong data”.

Graphical user interface, text

Description automatically generated

Figure 5. Example of objects created to be used in the data class. Testing out if the data class is working as desired. As you can see in the figure, when the wrong data type is entered, it is highlighted yellow. Python is indicating there might be an issue.

By testing out the data class section of the code, I know if the data class is working to catch the errors in the data inputs. When I run the above script, in Figure 5, I get errors since “Apple” can’t be converted into a float.

## File Processing Class

The next class I set-up was for processing the file. As in previous assignments, we already had the code for reading data from the file and then saving data to the file. As with other files, when we open the file, we go through the lines in the file, and split them up based on the “,” separating the product name and price. Once the row is created, append it to the list of rows. If the file is read ok and there are no issues, then the message error would be that the file was found and read.

Text

Description automatically generated

Figure 6. Example of the script used to read the data from the file and place it into a list of object rows.

The next function in the file processing section is the ability to save the data to file. Again, like in previous assignments, we already had the base script to save the data to the file. Since we are working with a list of objects, there are a few minor modifications that are needed.

Open the file in write mode, for each of the product objects write the product information to the file. We declared the string function in the data class for how we wanted to write the data, in this example it would be with a comma separation.

Text

Description automatically generated with low confidence

Figure 7. Example of the script to write the product name and product price with a comma separation.

Graphical user interface, text, application

Description automatically generated

Figure 8. Example of how to save the list of object rows to a file for use later. This script also shows an example of error handling that can be used if there is an error that comes up while saving the file.

## Presentation Class

The last of the classes is the presentation code. The presentation class is used to store the print statements for program. In this assignment we are using similar script to previous assignments. The main change for this section was for the data input from the user and printing the list of object rows.

This assignment asks for the input to go into an object row. With this difference there is a small change to how you save the data object to the list of object rows. The user inputs go through the data class to be checked as the product name and product price. Once through the data class, the object row is added to the list of products.

A screenshot of a computer

Description automatically generated with low confidence

Figure 9. Example of how to add an object row to a list of object rows. The data inputs are also run through the Product class to make sure the inputs follow the data class “rules”.

The last part of the presentation code that needs updating is the printing of the current list of products and prices. For each of the rows in the list of objects, print the name and string version of the price.

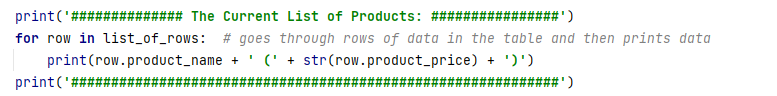


Figure 10. Example of the script used to print the rows in the list of objects.

## Testing of Code

I started the code with minor error handling inputs. I wanted to test out the code in order to figure out what specific error handling exceptions should be raised. When going through the main body of the script, the first function called is to read the data from the file. If the file doesn’t exist that would cause an error.

During testing, I also figured out there would be issues if the file didn’t have the product price as a string that could be converted into a float. If the starting file information was bad, then that would also cause an error.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 11. Example of the script for error handling when reading the file. If the file doesn’t exist an error will arise from FileNotFound. If there is an issue with the data in the file like a string can’t be converted into a float, it might raise a ValueError issue. A final general exception is used at the end to capture any other issues that arise.

The other place I added some extra error handling was with the user input for product name and product price. The product price was where I found most of the problems. If the data input couldn’t be converted to a float then it would cause an error, such as ValueError. I also added a general error exception to capture is there was an issues other than with the ValueError.

Graphical user interface, text, application

Description automatically generated

Figure 12. This is the example script for the error handling for if the user inputs “bad” data that doesn’t fit the “rules” set out in the data class. I know that there will be a ValueError if a float can’t be created. I used a general exception as well to catch any other issues.

# Summary

This module and assignment were to show how and when objects should be used. An object allows us to use the same functions/class to run through different data sets. In Assignment 8, we read data from a file into a list of objects, then added an object row to that list, and then saved the list of object rows to a file for use later. Much of the code was like previous assignments with a few twists when work with objects.

# Title

\*\*Dev:\*\* \*LTurner\*

\*\*Date:\*\* \*9.5.2021\*

## Introduction

This document will review information about classes. We have learned that classes are simply just groups of functions.

In this document we will review classes more in-depth, specifically looking at how to use objects and how to set-up a data class.

Objects allow us to use a class with many different data inputs.

Data classes allow us to do extra processing and check with data before we use it in the rest of the program.

## Classes

A class is simple a group of functions. The functions can do various things. It is best practice to separate the

script into classes based on their functions. A standard way to set-up functions by setting up three classes; data,

processing, and presentation.

When working with classes, you can work with them indirectly or directly.

### Object vs Class

When working with a class directly there is just one instance that the set of functions (class) can be stored at one time.

However, if we use the code indirectly, we can create many different instances using the same function.

We can use one class to do work with many different sets of data or inputs.

The way I thought about it was with cookies. I have a set of cookie cutters that is going to be my class.

The cookie cutters can be used over and over. Next, I want to make more than one set of cookies, but with different cookie doughs.

The cookies are all the different objects, different inputs to be used in all the different cookie cutters.

I found a few good additional resources to help define the differences between objects and classes.

The first one was from a site called LearnPython.org (External Source: [LearnPython](https://www.learnpython.org/en/Classes\_and\_Objects)). I really liked this source because it

stepped through using different inputs but getting different outputs. It really helped figure out the difference between

indirect and direct use of classes. ProgramWiz had some other great examples to set through how to set-up a class for indirect

use (External Source: [ProgramWiz](https://www.programiz.com/python-programming/class#:~:text=Python%20is%20an%20object%20oriented%20programming%20language.%20Unlike,a%20class%20is%20a%20blueprint%20for%20that%20object.)).

One key difference is the use of “self”. When using the class directly or in static mode, there is not need for the use

of “self”. When working indirectly with a class using objects, there is a need for self. You must tell Python that the object

is it’s self and not something else. Each object is itself, and Python must know that when working with objects.

### Setting Up a Class

A class is made up of a standard pattern. The pattern involves fields, constructors, properties, and methods.

![Figure 1.](/docs/Picture1.png "Figure 1")

Figure 1. Example of the standard pattern used for classes. A class includes, fields, constructor (attributes), properties,

and methods.

Fields in classes are just data, such as a product name or product price. Constructors are methods or functions,

that run when you create the object from the class. Normally a constructor is used to initialize the value from the field data.

Attributes are used as an imaginary or invisible field that is created. See below for an example of a constructor and attributes.

![Figure 2.](/docs/Picture2.png "Figure 2")

Figure 2. Example of how-to set-up a constructor and it’s attributes for product name and product price.

It is possible to raise an exception in the error handling process, but when this is done, it causes the program to stop.

Properties are functions that allow you to set different information about the field. The two different properties are a setter

and a getter. A setter allows for validation and error handling. A getter allows you to format the attribute or field. There is

a set code used when writing the script for a getter and setter, see below. The getter and setter for product name.

![Figure 3.](/docs/Picture3.png "Figure 3")

Figure 3 . Getter and setter code for the product name attribute.

Methods are just functions inside of a class. The functions can be printing a string to reading data to a file.

One important not for methods is to know if they are indirect or direct. As mentioned previously, indirect methods

require the use of self and direct methods must be defined as static method.

## Working through Assignment 08

The goal for this week assignment was to use indirect objects to create a list of product prices and product names.

As with previous assignments we also needed to be able to save the list to a file and then read the file back to the list.

I broke the script writing into four parts: data, processing, presentation, and main body.

### Data Class

As discussed in the previous section, the data class is the group of function used to process the data.

There are a few standard parts for a data class: fields, constructors and attributes, properties, and methods.

For the assignment, we needed to work with the product name and product price.

For product name, I knew that it needed to be a string. The product price needed to be a float.

The data class allows for the input values from the user to be checked to make sure that they are the types that we want.

We check the data types in the constructors/attributes, and in properties. Properties getters and setters are also another

place where the data is checked to make sure it fits what we need.

![Figure 4.](/docs/Picture4.png "Figure 4")

Figure 4. Example of the constructor in my data class for Assignment 08.

The data class is used when first reading the data from the file and when the user enters data into the program.

All the data must follow the rules lined out in the data class.

I tested the data class by creating objects for the code to use. In the testing process,

I used different data types to see what errors would come up from inputting the “wrong data”.

![Figure 5.](/docs/Picture5.png "Figure 5")

Figure 5. Example of objects created to be used in the data class. Testing out if the data class is working as desired.

As you can see in the figure, when the wrong data type is entered, it is highlighted yellow.

Python is indicating there might be an issue.

By testing out the data class section of the code, I know if the data class is working to catch the errors in the data inputs.

When I run the above script, in Figure 5, I get errors since “Apple” can’t be converted into a float.

### File Processing Class

The next class I set-up was for processing the file. As in previous assignments, we already had the code for reading data

from the file and then saving data to the file. As with other files, when we open the file, we go through the lines in the

file, and split them up based on the “,” separating the product name and price. Once the row is created, append it to the

list of rows. If the file is read ok and there are no issues, then the message error would be that the file was found and read.

![Figure 6.](/docs/Picture6.png "Figure 6")

Figure 6. Example of the script used to read the data from the file and place it into a list of object rows.

The next function in the file processing section is the ability to save the data to file. Again, like in previous assignments,

we already had the base script to save the data to the file. Since we are working with a list of objects, there are a few minor

modifications that are needed.

Open the file in write mode, for each of the product objects write the product information to the file.

We declared the string function in the data class for how we wanted to write the data, in this example it would be with a

comma separation.

![Figure 7.](/docs/Picture7.png "Figure 7")

Figure 7. Example of the script to write the product name and product price with a comma separation.

![Figure 8.](/docs/Picture8.png "Figure 8")

Figure 8. Example of how to save the list of object rows to a file for use later. This script also shows an example of error

handling that can be used if there is an error that comes up while saving the file.

### Presentation Class

The last of the classes is the presentation code. The presentation class is used to store the print statements for program.

In this assignment we are using similar script to previous assignments. The main change for this section was for the data input

from the user and printing the list of object rows.

This assignment asks for the input to go into an object row. With this difference there is a small change to how you save the

data object to the list of object rows. The user inputs go through the data class to be checked as the product name and product price. Once through the data class, the object row is added to the list of products.

![Figure 9.](/docs/Picture9.png "Figure 9")

Figure 9. Example of how to add an object row to a list of object rows. The data inputs are also run through the Product

class to make sure the inputs follow the data class “rules”.

The last part of the presentation code that needs updating is the printing of the current list of products and prices.

For each of the rows in the list of objects, print the name and string version of the price.

![Figure 10.](/docs/Picture10.png "Figure 10")

Figure 10. Example of the script used to print the rows in the list of objects.

### Testing of Code

I started the code with minor error handling inputs. I wanted to test out the code in order to figure out what specific error

handling exceptions should be raised. When going through the main body of the script, the first function called is to read the

data from the file. If the file doesn’t exist that would cause an error.

During testing, I also figured out there would be issues if the file didn’t have the product price as a string that could

be converted into a float. If the starting file information was bad, then that would also cause an error.

![Figure 11.](/docs/Picture11.png "Figure 11")

Figure 11. Example of the script for error handling when reading the file. If the file doesn’t exist an error will arise from

FileNotFound. If there is an issue with the data in the file like a string can’t be converted into a float, it might raise a

ValueError issue. A final general exception is used at the end to capture any other issues that arise.

The other place I added some extra error handling was with the user input for product name and product price. The product price

was where I found most of the problems. If the data input couldn’t be converted to a float then it would cause an error, such as

ValueError. I also added a general error exception to capture is there was an issues other than with the ValueError.

![Figure 12.](/docs/Picture12.png "Figure 12")

Figure 12. This is the example script for the error handling for if the user inputs “bad” data that doesn’t fit the “rules” set out

in the data class. I know that there will be a ValueError if a float can’t be created. I used a general exception as well to catch any other issues.

## Summary

This module and assignment were to show how and when objects should be used. An object allows us to use the same

functions/class to run through different data sets. In Assignment 8, we read data from a file into a list of objects,

then added an object row to that list, and then saved the list of object rows to a file for use later.

Much of the code was like previous assignments with a few twists when work with objects.